

Fonduer: Knowledge Base Construction from Richly Formatted Data

Sen Wu Luke Hsiao Xiao Cheng Braden Hancock
Theodoros Rekatsinas Philip Levis Christopher Ré
Stanford University

{senwu, lwhsiao, xiao, bradenjh, thodrek, pal, chrismre}@cs.stanford.edu

ABSTRACT

We introduce **Fonduer**, a knowledge base construction (KBC) framework for richly formatted information extraction (RFIE), where entity relations and attributes are conveyed via structural, tabular, visual, and textual expressions. **Fonduer** introduces a new programming model for KBC built around a unified data representation that accounts for three challenging characteristics of richly formatted data: (1) prevalent document-level relations, (2) multimodality, and (3) data variety. **Fonduer** is the first KBC system for richly formatted data and uses a human-in-the-loop paradigm for training machine learning systems, referred to as data programming. Data programming softens the burden of traditional supervision by only asking users to provide lightweight functions that programmatically assign (potentially noisy) labels to the input data. **Fonduer**'s unified data model, together with data programming, allows users to use domain expertise as weak signals of supervision that help guide the KBC process over richly formatted data. We evaluate **Fonduer** on four real-world applications over different domains and achieve an average improvement of 42 F1 points over the upper bound of state-of-the-art approaches. In some domains, our users have produced up to $1.87\times$ the number of correct entities compared to expert-curated public knowledge bases. **Fonduer** scales gracefully to millions of documents and is used in both academia and industry to create knowledge bases for real-world problems in many domains.

1. INTRODUCTION

Knowledge base construction is the process of populating a database with information from data such as text, tables, or video. Extensive efforts have been made to build large, high-quality knowledge bases (KBs), such as Freebase [4], YAGO [28], IBM Watson [5, 9], PharmGKB [14], and Google Knowledge Graph [27]. Traditionally, KBC solutions have focused on unstructured text, with great success [16, 19, 26, 31]. These KBC systems already support a broad range of downstream applications such as information retrieval, question answering, medical diagnosis, and data visualization. However, troves of information remain untapped in *richly formatted* data, where relations and attributes are expressed via combinations of textual, structural, tabular, and visual cues. In these scenarios, the semantics of the data are significantly affected by the organization and layout of the document. Examples of richly formatted data include webpages, business reports, product specifications, and scientific literature.

Transistor Datasheet

SMBT3904, MMBT3904				Font: Arial; Size: 12; Style: Bold
NPN Silicon Switching Transistors • High DC current gain: 0.1 mA to 100 mA • Low collector-emitter saturation voltage				From header
Maximum Ratings				
Parameter	Symbol	Value	Unit	
Collector-emitter voltage	V_{CEO}	40	V	
Collector-base voltage	V_{CBO}	60		
Emitter-base voltage	V_{EB0}	6		
Collector current		200	mA	Aligned
Total power dissipation	P_{tot}	330	mW	
$T_S \leq 60^\circ\text{C}$		250		
$T_S \leq 115^\circ\text{C}$				
Junction temperature	T_J	150	$^\circ\text{C}$	
Storage temperature	T_{stg}	-65 ... 150		

Knowledge Base

HasCollectorCurrent	
Transistor Part	Current
SMBT3904	200mA
MMBT3904	200mA

Figure 1: An example of KBC for the HasCollectorCurrent(Transistor Part, Current) relation from transistor datasheets. The Transistor Part candidate mentions are boxed in blue, while Current candidate mentions are boxed in green.

Example 1.1 (HasCollectorCurrent). We illustrate RFIE with a running example from the ELECTRONICS domain (further introduced in Section 5.1). Transistors are semiconductor devices used as switches or amplifiers. Their electrical specifications are published by manufacturers in datasheets like the one shown in Figure 1. The goal of this task is to use transistor datasheets to build a KB of transistors' maximum collector current. This KB could be used, for example, to build a tool that verifies that transistors do not exceed their maximum ratings in a circuit.

Challenges. KBC on richly formatted data poses a number of challenges beyond those present with unstructured data: (1) accommodating *prevalent document-level relations*, (2) capturing the *multiplicity* of information in the input data, and (3) addressing the tremendous *data variety*.

Prevalent Document-Level Relations We define context as the information considered when extracting a relation, ranging from a single sentence to a whole document. KBC systems typically limit context scope to a few sentences or a single table, assuming that relations are expressed relatively locally. However, for richly formatted data, many relations rely on information from throughout the entire document to be properly extracted.

Example 1.2 (Document-Level Relations). In Figure 1, transistor parts are located in the document header (boxed in blue), and the collector current value is in a table cell (boxed in green). Moreover, the interpretation of some numerical values depends on their units expressed in another table column (e.g., 200 mA).

Limiting the context scope to a single sentence or a single table misses many potential entries, up to 97% in the ELECTRONICS application, for example. On the other hand, naïvely considering all possible entity pairs throughout the document as candidates renders the extraction problem unnecessarily computationally intractable due to the combinatorial explosion of candidates.

Multimodality Classical KBC systems usually model input data as unstructured text [16, 18, 26]. With richly formatted data, however, semantics are also expressed through multiple modalities—textual, structural, tabular, and visual.

Example 1.3 (Multimodality). *In Figure 1, important information, such as headers, is expressed in larger, bold fonts (displayed in yellow), and the meanings of table cell contents depend heavily on the content with which they are visually aligned (shown by the red arrow).*

Signals from each modality in richly formatted data can vary wildly, and can be challenging to utilize consistently.

Data Variety With richly formatted data, the same information can be presented in many different formats and styles, in addition to linguistic variations. For example, tabular data can be oriented vertically or horizontally and numerical values have many formats:

Example 1.4 (Data Variety). *In Figure 1, numeric intervals are expressed as “-65 ... 150”, but other datasheets show intervals as “-65 ~ 150”, or “-65 to 150”. Similarly, attribute names are synonymous with symbols, like “Collector current” and “I_c”.*

Furthermore, at a macro level, data variety over time or across domains requires KBC systems to be generalizable and robust against changing input data.

Our Approach. We introduce **Fonduer**, the first KBC system for richly formatted information extraction. To account for the variability of richly formatted data, **Fonduer** automatically encodes documents of heterogeneous formats into a new *unified data model* that preserves structural and semantic information across different modalities of the data.

Given a collection of richly formatted documents, **Fonduer** uses statistical learning and inference to construct a knowledge base with facts from the input documents. **Fonduer** operates upon (1) *candidates*, which are potential entries in a target KB; (2) *features*, which describe each candidate and serve as evidence for a *classifier* to distinguish between true and false candidates; and (3) *training data*, (i.e., ground truth information on a subset of candidates, that is used to train the aforementioned classifier) [26]. **Fonduer**’s unified data model provides a formalism for specifying candidates, features, and training data over richly formatted data.

Fonduer uses a new human-in-the-loop paradigm for training machine learning systems called *data programming* [25]. Data programming softens the burden of traditional supervision by only asking users to provide lightweight user-defined functions that programmatically assign (potentially noisy) labels to the input data. Data programming uses a generative probabilistic model to denoise the labeled data and train the final machine learning model.

We show how **Fonduer**’s unified data model together with data programming enable users to define labeling functions that consider different modalities of richly formatted data. We find that across four real-world applications, **Fonduer** en-

ables users to build high-quality KBs, achieving an average improvement of 42 F1 points over state-of-the-art systems.

Technical Challenges Using statistical learning and inference for KBC over richly formatted data raises several technical challenges:

First, the increased breadth of signals and context scope in richly formatted data magnifies the need for large amounts of training data. Manual annotation can be prohibitively expensive, especially when domain expertise is required. At the same time, human-curated KBs, which can be used to generate training data, may exhibit low coverage or not exist altogether. Alternatively, weak supervision sources can be used to programmatically create large training sets, but it is often unclear how to consistently apply these sources to richly formatted data. Whereas patterns in unstructured data can be identified based on text alone, expressing patterns consistently in richly formatted data is challenging.

Second, in richly formatted data, the entity mentions that form potential relation mentions often come from different contexts or even different context types (e.g., text and tables), which can be far apart in the document. Reasoning about candidates that are not locally adjacent leads to a combinatorial explosion of possible candidates, and thus random variables, which need to be considered during learning and inference. This leads to a fundamental tension between building a practical KBC system and learning accurate models that exhibit high recall. In addition, the combinatorial explosion of possible candidates results in a large class imbalance, where the number of “True” candidates is significantly smaller than the number of “False” candidates. Therefore, techniques to prune candidates to balance running time and end-to-end quality are required.

Finally, the variety and multimodality of richly formatted data make the feature space of KBC candidates even larger and sparser than that of unstructured text. Moreover, the variety limits the coverage and accuracy of any individual heuristics used for supervision, requiring multiple supervision sources to be combined in order to sufficiently cover all the relations expressed in a corpus.

Technical Contributions

- We are the first to use data programming for KBC over richly formatted data. We show how **Fonduer**’s unified data model enables users to define labeling functions that consider different modalities of richly formatted data and demonstrate that complex real-world classifiers can be learned from noisy labels (see Section 3).
- We present a series of optimizations based on the data access patterns for different phases of KBC workloads and perform an empirical analysis on the effect of different storage backends on the end-to-end runtime of KBC. We exploit the immutability of candidates, features, and labels in our system to achieve up to 1000× speed up compared to traditional implementations of KBC systems. This allows users to iteratively improve KBC quality on millions of candidates (see Section 4).
- We evaluate **Fonduer** by performing cold-start KBC in four distinct domains and achieve an average improvement of 42 F1 points over the upper bound of state-of-the-art techniques. We show how candidate generation, featurization, supervision and learning affects end-to-end quality (see Section 5).

We validate our framework on ELECTRONICS, ADVERTISEMENTS, PALEONTOLOGY, and GENOMICS domains (further described in Section 5.1) and produce KBs which are used in production in each domain. These KBs represent over 151GB of data and over 9.6M documents from webpages, literature, and technical datasheets.

Our evaluation shows that **Fonduer** efficiently extracts difficult relations that rely on document-wide signals and achieve an improvement of up to 70 F1 points ($12.7\times$) over traditional approaches which utilize only sentence-level contexts. We also show via a feature ablation study that including features for each modality yields higher quality in all domains and see quality deteriorate up to 33 F1 points when features from a single modality are removed. By using a noise-aware learning framework, we improve end-to-end quality up to 24 F1 points over techniques that do not model noise in supervision sources.

Real-world Impact. **Fonduer** is already being used in both academia and industry to create KBs to tackle real-world problems in many domains. **Fonduer** has been used to create KBs for DARPA MEMEX (from 106M web pages and 0.5M forums posts across 1000s of web domains), which is used daily by law enforcement and has helped make important arrests. Electrical engineering researchers are creating KBs of over 5M electronic components for applications like design tools and embedded device generation. Paleontologists have built KBs over a few weeks with quality comparable to the Paleobiology Database, which contains hundreds of thousands of entries manually populated over 9 continuous years [23]. Efforts are being made to create manually-curated databases of genome-wide association studies [3,30], but they are incomplete and still vary greatly in their scope. Our users have built GwasDB¹, a machine reading system for automatically extracting those associations from scientific literature with comparable quality to public databases.

2. BACKGROUND

We introduce several key definitions and provide an overview of the existing techniques that are utilized by **Fonduer**.

2.1 Knowledge Base Construction

The input to a KBC system is a collection of documents. The output of the system is a relational database containing facts extracted from the input and stored in an appropriate schema. To describe the KBC process, we adopt the standard terminology from the KBC community². There are four types of objects that play integral roles in KBC systems: (1) *entities*, (2) *relations*, (3) *mentions* of entities, and (4) *relations mentions*.

An **entity** e in a knowledge base corresponds to a distinct real-world person, place, or object. Entities in a knowledge base can be grouped in different **entity types** T_1, T_2, \dots, T_n . Entities in a knowledge base participate in relationships. A relationship between n entities is represented as an n -ary **relation** $R(e_1, e_2, \dots, e_n)$ and is described by a **schema** $S_R(T_1, T_2, \dots, T_n)$ where $e_i \in T_i$. A **mention** m is a span of text that refers to an entity. KBC systems typically assume that all mentions of entities in a document have a

corresponding span of text that refers to them. A relation mention **candidate** is a particular n -ary tuple $c = (m_1, m_2, \dots, m_n)$ that represents an instance of a relation $R(e_1, e_2, \dots, e_n)$ in a document. If a candidate is classified as true, it is called a **relation mention**, r_R .

Example 2.1 (KBC). *Consider the HasCollectorCurrent task in Figure 1. **Fonduer** takes a corpus of transistor datasheets as input and constructs a KB containing the (transistor part, current) binary **relation** as output. Parts like SMBT3904 and currents like 200mA are **entities**. The spans of text that read “SMBT3904” and “200” (boxed in blue and green, respectively) are **mentions** of those two entities, and together they form a **candidate**. If the evidence in the document suggests that these two mentions are related, then the output relation table should include the **relation mention** (SMBT3904, 200mA).*

The KBC problem can be formally defined as follows:

Definition 2.1 (Knowledge Base Construction).

Given a set of documents D and a knowledge base schema $S_R(T_1, T_2, \dots, T_n)$, where each T_i corresponds to an entity type, extract a set of relations r_R from D , which populate the schema’s relational tables.

Like other trained KBC systems [6,26], **Fonduer** converts KBC to a statistical learning and inference problem: each candidate is assigned a Boolean random variable that can take the value “True” if the corresponding relation mention is correct, or “False” otherwise. In trained KBC systems, each candidate is associated with certain features that provide evidence on the value that the corresponding random variable should take. Trained KBC systems use machine learning to maximize the probability of correctly classifying candidates, given their features and ground truth examples.

2.2 Data Programming

Trained KBC systems rely heavily on ground truth data (called *training data*) to achieve high quality. Traditionally, manual annotations or incomplete KBs are used to construct training data for trained KBC systems. However, these resources are either costly to obtain or may have limited coverage over the candidates considered during the KBC process. To address this challenge, **Fonduer** builds upon the newly introduced paradigm of data programming (DP) [25], which enables domain experts to programmatically generate large training data sets by leveraging multiple weak supervision sources and domain knowledge.

Data programming provides a simple, unifying framework for weak supervision, in which training labels are noisy and may come from multiple, potentially overlapping sources. In data programming, users encode this weak supervision in the form of *labeling functions*, which are user-defined functions that each provide a label for some subset of the data, and collectively generate a large and potentially overlapping set of training labels. Many different weak supervision approaches can be expressed as labeling functions. This includes strategies which utilize existing knowledge bases, individual annotator’s labels (as in crowdsourcing), or user-defined functions that rely on domain-specific patterns and dictionaries to assign labels to the input data.

The aforementioned sources of supervision can have varying degrees of accuracy, and may conflict with each other. Data programming relies on a generative probabilistic model to estimate the accuracy of each labeling function by reason-

¹<https://github.com/kuleshov/gwasdb>

²<http://www.itl.nist.gov/iad/mig/tests/ace/>

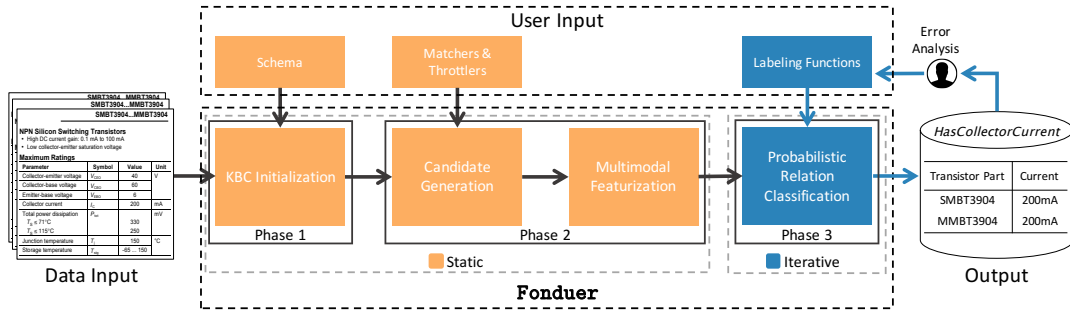


Figure 2: An overview of Fonduer KBC over richly formatted data. Given a set of richly formatted documents and a series of lightweight inputs from the user, Fonduer extracts facts and stores them in a relational database.

ing about the conflicts and overlap across labeling functions. The estimated labeling function accuracies are in turn used to assign a probabilistic label to each candidate, which are used in conjunction with a noise-aware discriminative model to train a machine learning model for KBC.

2.2.1 Components of Data Programming

The main components in data programming are as follows:

Candidates A set of candidates C are probabilistically classified.

Features Features are real numbers characterizing each candidate. A list of features $f = \Phi(c)$ is generated by applying feature extractor Φ on candidate c . After featurizing k candidates, each with a list of n features, we have a feature matrix $X \in \mathbb{R}^{k \times n}$.

Labeling Functions Labeling functions are used to programmatically provide labels for training data. A labeling function is a user-defined procedure that takes a candidate as input and outputs a label. Labels can be as simple as true or false for binary tasks, or one of many classes for more complex multiclass tasks. Since each labeling function is applied to all candidates and labeling functions are rarely perfectly accurate, there may be disagreements between them. The labeling functions provided by the user for binary classification can be more formally defined as follows: For each labeling function λ_i and $r \in C$, we have $\lambda_i : r \mapsto \{-1, 0, 1\}$ where $+1$ or -1 denotes a candidate as true or false, and 0 abstains. The output of applying a set of l labeling functions to k candidates is the label matrix $\Lambda \in \{-1, 0, 1\}^{k \times l}$.

Output Data programming frameworks output a confidence value p for the classification for each candidate as a vector $Y \in \{p\}^k$.

To perform data programming in **Fonduer**, we rely on a data programming engine, Snorkel³. Snorkel accepts candidates, features, and labels as input, and produces marginal probabilities for each candidate as output. These input and output components are stored as relational tables, whose schemas are detailed in Section 3.

2.2.2 Theoretical Guarantees

While data programming uses labeling functions to generate noisy training data, it theoretically achieves a learning rate similar to methods that use manually labeled data [25]. In the typical supervised learning setup, users are required

to manually label $\tilde{O}(\epsilon^{-2})$ examples for the target model to achieve an expected loss of ϵ . To achieve this rate, data programming only requires the user to specify a constant number of labeling functions that does not depend on ϵ . Let β be the minimum coverage across labeling functions (i.e., the probability that a labeling function provides a label for an input point), and γ be the minimum reliability of labeling functions, where $\gamma = 2 \cdot \alpha - 1$ with α denoting the accuracy of a labeling function. Then under the assumptions that: (1) labeling functions are conditionally independent given the true labels of input data, (2) the number of user-provided labeling functions is at least $\tilde{O}(\gamma^{-3}\beta^{-1})$, and (3) there are $k = \tilde{O}(\epsilon^{-2})$ candidates, data programming achieves an expected loss ϵ . Despite the strict assumptions with respect to labeling functions, we find that using data programming to develop KBC systems for richly formatted data leads to high-quality KBs (across diverse real-world applications) even when some of the data programming assumptions are not met (see Section 3.3 and Section 5.4.3).

3. KBC USING Fonduer

Given a set of documents, **Fonduer** decomposes KBC into three phases: (1) KBC initialization, (2) candidate generation and multimodal featurization, and (3) probabilistic relation classification. To perform probabilistic relation classification, **Fonduer** relies on the data programming paradigm (see Section 2.2). **Fonduer** adopts a “human-in-the-loop” approach which requires that users provide a series of lightweight inputs to guide the KBC process. An overview of **Fonduer** is shown in Figure 2. In this section, we describe the user inputs required by **Fonduer**, review the data pipeline, then comment on common modes of operation.

3.1 User Inputs

Fonduer assumes that the user provides certain domain-specific inputs that help guide the KBC process: (1) a *schema* to materialize a database used to store all extracted relations between entities, (2) a collection of user-defined functions, referred to as *matchers* and *throttlers*, that are used to drive candidate generation, and (3) a set of *labeling functions*, which capture the domain expertise of the user and are integral in using data programming for probabilistic relation classification. We review each of these inputs in turn.

Schema First, the user provides a schema $S_R(T_1, \dots, T_n)$, which defines a relation R to be extracted from the input documents. This schema is used by **Fonduer** to initialize the relational tables that store all extracted relation mentions. An example of such a schema is provided below.

³<http://hazyresearch.github.io/snorkel/>

Example 3.1 (Relation Schema). *An example SQL schema for the relation in Figure 1 is:*

```
CREATE TABLE HasCollectorCurrent(
  TransistorPart varchar,
  Current varchar);
```

Matchers To generate candidates for a user-defined relation R , **Fonduer** asks users to define *matchers* for all distinct mention types in schema S_R . In **Fonduer**, matchers are Python functions which accept as input a span of text, and output whether or not the match conditions were met. Matchers can refer to characteristics from any modality.

Example 3.2 (Matchers). *From the HasCollectorCurrent relation in Figure 1, users define matchers for each type of the schema. A dictionary of valid transistor parts can be used as the first matcher. For the current, users can exploit the pattern that current values are commonly expressed as a numerical value in the interval 100 and 995 for their second matcher.*

```
# Use a dictionary to match transistor parts
def transistor_part_matcher(s):
    return 1 if s in part_dictionary else 0

# Use RegEx to extract numbers between [100, 995]
def max_current_matcher(s):
    return 1 if re.match('[1-9][0-9][0-5]', s) else 0
```

Throttlers Users can optionally provide *throttlers*, which act as hard filtering rules to reduce the number of candidates that are materialized. Throttlers are also Python functions, but rather than accepting spans of text as input, they operate on candidates, and output whether or not a candidate meets the specified condition.

Example 3.3 (Throttler). *Continuing the example shown in Figure 2, the user provides a throttler which only keeps currents that have “Value” as a column header.*

```
def value_in_column_header(c):
    if 'Value' in header_ngrams(c.current):
        return 1
    else:
        return 0
```

Labeling Functions To generate training data for probabilistic relation classification, users provide **Fonduer** with *labeling functions* (LFs). These provide weak supervision in the form of Python functions that take a candidate as input and assign +1 to label a candidate as true, 0 to abstain, or -1 to label a candidate as false.

Example 3.4 (Labeling Functions). *Looking at the data-sheet in Figure 1, users can express patterns such as having the part and current y-aligned on the visual rendering of the page. Similarly, users could write an intuitive rule that labels a candidate as true if it has the word “current” in the same row as the candidate current mention.*

```
# Rule-based LF based on visual information
def y_axis_aligned(c):
    return 1 if c.part.y == c.current.y else 0

# Rule-based LF based on tabular content
def has_current_in_row(c):
    if 'current' in row_ngrams(c.current):
        return 1
    else:
        return 0
```

We find that a only a small number of labeling functions are needed to achieve high quality KBC. For example, in the **ELECTRONICS** application, we find that, on average, 16 labeling functions are sufficient to achieve an average F1 score of over 75 F1 points.

3.2 Data Pipeline

Fonduer uses the previously desribed inputs to form a data programming problem and construct the final KB.

KBC Initialization This is the first phase in **Fonduer**’s pipeline. During this phase, **Fonduer** uses the user-specified *schema* to initialize a relational database where the output KB will be stored. Furthermore, **Fonduer** iterates over its input *corpus* and transforms each document with the unified data model, capturing the variability and multimodality of richly formatted documents. This unified data model serves as an intermediate representation used to generate candidates and features in the next phase. **Fonduer**’s unified data model is shown in Figure 3, where each node in the directed acyclic graph represents a *context*, such as a sentence or table, with corresponding attributes and pointers (see Section 4.1 for details). The unified data model allows **Fonduer** to traverse a document and access modality information. Users can easily query their data using consistent syntax regardless of the original input format.

Candidate Generation and Multimodal Featurization Given the unified data model from Phase 1, **Fonduer** extracts relations candidates based on the user-provided *matchers* and *throttlers*. **Fonduer** traverses each instance of the unified data model to extract candidates using user-provided matchers. By applying matchers to each leaf of the unified data model, **Fonduer** can generate sets of mentions for each component of the schema. The cross-product of these mentions produce candidates:

Candidate($id_{\text{candidate}}$, $mention_1$, ..., $mention_n$)

where mentions are spans of text and contain pointers to their context in the unified data model of their document.

While matchers allow users to reduce the number of *entity mentions* considered, users can also specify throttlers to further reduce the number of *candidates* before they are materialized. By using matchers and throttlers, users can intelligently reduce the number of candidates materialized by over 100×. For example, in Figure 1, a naïve cross product of all words and numbers results in 800 candidates, while using regular expressions as matchers can reduce the number of candidates to less than 5 while maintaining perfect recall. In Section 4.2, we analyze the effects of pruning candidates.

As the final step in this phase, **Fonduer** featurizes each candidate with a feature set from the textual, structural, tabular, and visual modalities of the data.

Features($id_{\text{candidate}}$, $feature_set$)

After this phase, **Fonduer** has generated the set of candidates C and the feature matrix \mathcal{X} discussed in Section 2.2.

Note that Phase 1 and 2 are relatively static and typically only executed once during the KBC process. See Section 4.5.2 for analysis of how this pattern affects the implementation of **Fonduer**.

Probabilistic Relation Classification In Phase 3, *Fonduer* applies the user’s *labeling functions* to each of the candidates to create a label matrix Λ . These labels, along with the candidates and features from Phase 2, are passed to a data programming engine, which performs noise-aware learning and inference to determine how to classify each candidate.

Labeling functions can encode supervision from a variety of sources—domain expert heuristics, crowdsourcing, or distant supervision from an existing but possibly incomplete knowledge base. Furthermore, labeling functions also allow users to label correctness based on any modality of the data. See Section 4.4 for details on the importance of including many supervision sources.

We find that a small number of labeling functions can achieve high quality KBC. For example, over our four domains, an average of 10 labeling functions are sufficient.

Each candidate is assigned a set of labels (one from each labeling function), which together form a label matrix Λ .

Labels(id_{candidate}, label_{set})

The label matrix, along with the set of candidates C and feature matrix X , are input to the data programming engine. *Fonduer* assigns a marginal probability for each candidate and provides debugging information to facilitate error analysis, which can be used in the iterative KBC process. Users can specify a *threshold* to classify these candidates based on the requirements of their application.

Note that unlike Phase 2, Phase 3 supports user interaction to allow for iteratively improving and adjusting labeling functions after error analysis. Section 4.5.2 presents analysis of how the iterative nature of this phase is reflected in the design space tradeoffs.

3.3 KBC Development

The design of *Fonduer* was strongly guided by interactions with our collaborators. We found that in practice, the data programming paradigm dictates a new approach for KBC development. Previously, development focused on feature engineering, with each modification requiring validation in the form of rerunning the feature extraction, learning, and inference stages. With data programming, however, features are generated automatically and the emphasis is placed instead on supervision in the form of labeling functions. This results in two natural modes of operation for *Fonduer* applications: (1) development, and (2) production. During development, labeling functions are iteratively improved as they are applied to a small sample of labeled candidates and evaluated by the user on their accuracy and coverage (the fraction of candidates receiving non-zero labels). In practice, we found that approximately 20 iterations were sufficient for our users to generate a sufficiently tuned set of labeling functions. Then in production mode, the labeling functions are applied to the entire set of candidates and learning and inference are performed to create the desired KB.

Table 1: Average labeling function statistics over four domains.

	#LFs	Accuracy	Coverage
AVERAGE	10	0.73	0.54

The conditions under which data programming is guaranteed to have the same asymptotic scaling as methods which utilize labeled data are outlined in Section 2.2.2. While these conditions are relatively strict, we find empirically that

excellent results are achievable even when these conditions are relaxed. Table 1 shows the average number, accuracy, and coverage of the labeling functions used in each of our four applications. We see that users tend to write labeling functions with greater than 50% accuracy and coverage on average, and observed that our users’ labeling functions were not necessarily conditionally independent. Nevertheless, *Fonduer* achieves better than state-of-the-art quality in each domain. A detailed empirical study of how labeling functions and candidates affect the end quality of KBC output is shown in Section 5.4.3.

4. KBC FROM RF DATA

Richly formatted (RF) data exhibits incredible variation in terms of both the format variety of documents and the multimodality of richly formatted data. Furthermore, unlike KBC with unstructured text, richly formatted data requires the ability to form candidates from document-level context scopes. We detail how *Fonduer* addresses these challenges by first describing the unified data model, which plays a key role in each phase of KBC using *Fonduer*. We then present key design choices for the primary tasks of KBC—candidate generation, feature extraction, and supervision—while highlighting the technical contributions required for richly formatted data. We also provide empirical analysis of optimizations and tradeoffs of the implementation of *Fonduer*.

4.1 Unified Data Model

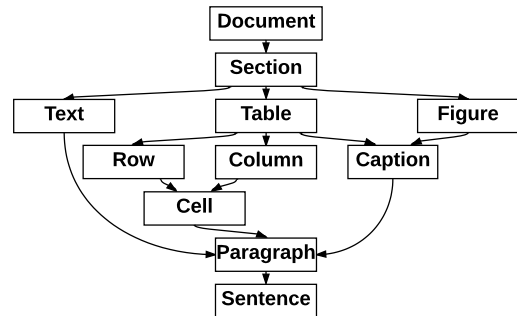


Figure 3: Fonduer’s unified data model.

First, we describe how richly formatted data can be effectively encoded in a unified data model for candidate generation, multimodal featurization, and supervision. To support KBC from richly formatted data, the model must:

- Generically capture documents of diverse types ranging from PDFs to HTML to XML in a unified manner.
- Preserve attributes from all modalities of the document since they contain valuable semantic information.
- Serve as an abstraction for system and user interaction.

Fonduer’s unified data model is a directed acyclic graph (DAG) that contains a hierarchy of contexts (represented as boxes in Figure 3), whose structure reflects the intuitive hierarchy of document components. In this graph, each node is a *context*. The root of the DAG is a *Document*, which contains *Section* contexts. Each section is divided into: *Texts*, *Tables*, and *Figures*. *Texts* can contain multiple *Paragraphs*; *Tables* and *Figures* can contain *Captions*; *Tables* can also contain *Rows* and *Columns*, which are in turn made up of *Cells*. Each context ultimately breaks down into *Paragraphs*

that are parsed into *Sentences*. In Figure 3, a downward edge indicates a parent-contains-child relationship.

In addition, for each context, we store the textual contents, pointers to the parent contexts, and a wide range of attributes from each modality found in the original document. For example, standard NLP pre-processing tools are used to generate linguistic attributes such as lemmas, part of speech (POS) tags, named entity recognition (NER) tags, dependency paths, etc. for each *Sentence*. Structural and tabular attributes of a *Sentence*, such as tags, and row/column information, and parent attributes, can be easily captured while traversing its path. Visual attributes for the document are recorded by storing bounding box and page information, if applicable, for each word in a *Sentence*.

Example 4.1 (Unified Data Model). *This stage takes raw documents as input and outputs their corresponding unified data models. In Figure 1, the data model representing the PDF would contain one Section with three children: a Text for the document header, a Text for the table description, and a Table for the table itself (made of 10 Rows and 4 Columns where each Cell links to both a Row and Column). The Text and Cell contexts would then contain Paragraphs, then Sentences.*

To construct the DAG for each document, we first extract all the words in their original order. For structural and tabular information we use tools such as Poppler⁴ to convert input file into HTML format; for visual information such as coordinates and bounding box we use a PDF printer to convert input file into PDF format. Lastly, if a conversion occurred, we associate the multimodal information in the converted file with all the extracted words. We align the word sequences of the converted file with their originals by checking if both their characters and number of repeated occurrences before the current word are the same. **Fonduer** can recover from conversion errors by relying on similar signals from other modalities.

Design Takeaways. **Fonduer** consolidates multiple document formats, multiple common types of contexts, and multiple modality semantics into a single model. This unified data model serves as the formal representation of the data that is utilized in all future stages of the extraction process.

4.2 Candidate Generation from RF Data

Candidate Generation from richly formatted data relies on access to document-level contexts, which is provided by **Fonduer**’s unified data model. Due to the significantly increased context needed for KBC from richly formatted data, naïvely materializing all possible candidates is intractable as the number of candidates grows combinatorially with the number of relation arguments. This combinatorial explosion can lead to performance issues for KBC systems. For example, in the **ELECTRONICS** domain, just 100 documents can generate over 1M candidates. In addition, we find that the majority of these candidates do not express true relations, creating a significant class imbalance.

To address this combinatorial explosion, users should use throttlers, in addition to matchers, to prune away excess candidates. Throttlers should:

- Keep high accuracy by only filtering negative candidates.

- Seek high coverage of the candidates.

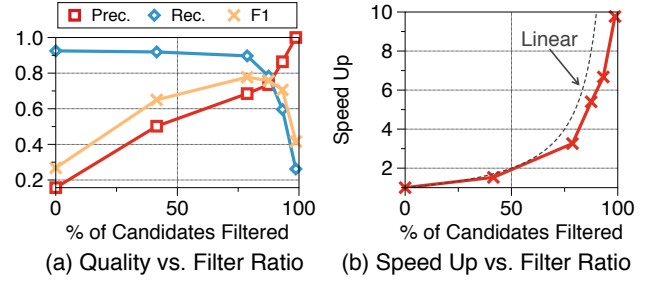


Figure 4: Tradeoff between (a) quality metrics and (b) execution time when pruning the number of candidates using throttlers in candidate generation.

Figure 4 shows the effect of throttler on both quality and performance in **ELECTRONICS** domain. We see that throttling significantly improves system performance. However, increased throttling does not monotonically improve quality. This tradeoff captures the fundamental tension between optimizing for system performance and optimizing for end-to-end quality. When no candidates are pruned, the class imbalance resulting from many negative candidates to the relatively small number of positive candidates devastates quality. Thus, as a rule of thumb, we recommend that throttlers be used to remove as many candidates as possible (in this case, approximately 75%) while maintaining high recall. Once recall begins to drop, the benefits of increased precision and class balance are outweighed.

Design Takeaways. The unified data model is necessary to perform candidate generation with richly formatted data. We find that in general, an ideal throttler is one that minimizes the size of the candidate set (for the sake of performance) while maximizing recall (for the sake of quality).

4.3 Multimodal Featurization

In KBC from richly formatted data, a variety of features that capture the different data modalities of the input provide strong evidence for the truthfulness of a candidate. **Fonduer** provides a feature library which captures a simple set of textual, structural, tabular, and visual data modalities by leveraging the unified data model of each input document. We find that this library is sufficient to achieve high quality without the need for user input, and provide examples in Figure 5. We discuss each of these modalities and how the corresponding features are generated.

Textual features These are traditionally the primary source of signal for relation extraction. These features assist with understanding words in a sentence using standard natural language processing techniques. For mention candidates, **Fonduer** include features such as words, bigrams, lemmas, parts of speech (POS), and named entity recognition (NER) tags (shown in blue in Figure 5). NER tags, for example, are useful in the **ELECTRONICS** domain where more than five instances of NER-tagged numbers in a table row often indicates a false candidate. For relation candidates, **Fonduer** also includes additional features that describe the relation as a whole, like the dependency parse path between relation arguments found in the same sentence.

⁴<https://poppler.freedesktop.org/>

POS: NN

Font: Arial; Size: 12; Style: Bold; SMBT3904, MMBT3904

NPN Silicon Switching Transistors

- High DC current gain: 0.1 mA to 100 mA
- Low collector-emitter saturation voltage

Maximum Ratings

Parameter	Symbol	Value	Unit
Collector-emitter voltage	V_{CEO}	40	V
Collector-base voltage	V_{CBO}	60	V
Emitter-base voltage	V_{EB}	60	V
Collector current		200	

POS: CD; NER: Num

Figure 5: An example of multimodal features for candidate (SMBT3904, 200) in Figure 1.

Structural features These provide signals intrinsic to a document’s structure. This feature type allows **Fondue** to learn from structural attributes such as parent and sibling relationships or recognizing that a mention candidate is within particular XML/HTML tags. In addition, structural features capture tag attributes such as font sizes and styles, shown in yellow in Figure 5, which are often used to highlight important keywords or information. **Fondue** extracts many structural features from the parents and siblings of mentions inside the unified data model of the document. **Fondue** also tracks the structural distance of relations, which helps when the mentions are visually distant, but structurally close together. As an example, featurizing a candidate with the lowest common ancestor in the unified data model is a positive signal for linking table captions to table contents.

Tabular features These are a special subset of structural features since tables are very common structures inside documents and have high information density. Table features are drawn from the grid-like representation of rows and columns stored in the unified data model, shown in green in Figure 5. In addition to the tabular location of mentions, **Fondue** also featurizes relations with special signals such as being in the same row or column. Consider, for example, a table that contains cells with multiple lines of text; recording that two entity mentions are in the same row captures a signal that a visual alignment feature could easily miss.

Visual features These provide signals observed from a visual rendering of a document. In cases where tabular or structural features are noisy—including nearly all documents converted from PDF to HTML by generic tools—visual features can provide a complementary view of the dependencies among text. Visual features encode many highly-predictive types of semantic information implicitly, such as position on a page, which may imply when text is a title or header. An example of this is shown in red in Figure 5.

Through a feature ablation study in Section 5.3.2, we show that using all feature types results in higher quality scores for each of our evaluation datasets, and find that quality can deteriorate up to 33 F1 points when a single feature type is removed. This feature library provides users a baseline set of features from each modality and decouples the development of features from RFIE.

Design Takeaways. In order to achieve high quality KBC with richly formatted data, it is vital to have features from

multiple data modalities. These features are only obtainable through traversing and accessing modality attributes stored in the unified data model.

4.4 Multimodal Supervision

Unlike KBC from unstructured text, KBC from richly formatted data requires supervision from multiple modality characteristics of the data. In richly formatted data, useful patterns for KBC are more sparse and hidden in non-textual signals, which motivates the need to exploit overlap and repetition in a variety of patterns over multiple modalities. **Fondue**’s unified data model allows users to directly express correctness using textual, structural, tabular, or visual characteristics, in addition to traditional supervision sources like existing KBs. In the **ELECTRONICS** domain, over 70% of labeling functions written by our users are based on non-textual signals. It is acceptable for these labeling functions to be noisy and conflict with one another. Data programming theory (Section 2.2.2) shows that with a sufficient number of labeling functions, data programming still can achieve quality comparable to using labeled data.

In Section 5.4.1, we find that using metadata, such as structural, tabular, and visual cues, results in an increase of 66 F1 points over using textual supervision sources alone. Using both sources gives a further increase of 2 F1 points over the 66 F1 improvement of metadata alone. Although dramatic improvement in quality is due to the characteristics of the **ELECTRONICS** domain, we also show that supervision using information from all modalities, rather than textual information alone, results in an increase of 43 F1 points on average over a variety of domains. Using multiple supervision sources is crucial to achieving high quality RFIE. Furthermore, with this approach, the size of the training set K scales with the amount of unlabeled input data directly, which is validated in Section 5.4.3.

Design Takeaways. Supervision using multiple modalities of richly formatted data is key to achieving high end-to-end quality. Like multimodal featurization, multimodal supervision is also enabled by **Fondue**’s unified data model.

4.5 System Design Considerations

We discuss design decisions for KBC from richly formatted data by analyzing the benefits of caching, as well as discussing data representation choices with respect to performance for the abstract data structures used in **Fondue**.

4.5.1 Data Caching

With richly formatted data, which frequently requires document-level context, thousands of candidates need to be featurized for each document. Candidate features are computed at both the mention level and relation level, by traversing the unified data model accessing modality attributes. Because each mention of a relation is part of many candidates, naïve featurization of candidates can result in the redundant computation of thousands of mention features. This pattern highlights the value of data caching when performing multimodal featurization on richly formatted data.

Traditional KBC systems that operate on single sentences of unstructured text pragmatically assume that only a small number of candidates will need to be featurized for each sentence, and do not cache mention features as a result.

Example 4.2 (Inefficient Featurization). In Figure 1, the transistor part mention MMBT3904 could be matched with up to 15 different numerical values in the datasheet. Without caching, the features of the MMBT3904 would be unnecessarily recalculated 14 times, once for each candidate.

In Example 4.2, eliminating unnecessary feature computations can improve performance by an order of magnitude.

To optimize the feature generation process, **Fonduer** implements a document-level caching scheme for mention features. The first computation of a mention feature requires traversing the unified data model. Then, the result is cached for fast access if the feature is needed again. All features are cached until all candidates in a document are fully featurized, after which the cache is flushed. Because **Fonduer** operates on documents atomically, caching a single document at a time improves performance without adding significant memory overhead. In the ELECTRONICS domain, we find that caching achieves over 100 \times speed up on average and in some cases even over 1000 \times , while only accounting for approximately 10% of the memory footprint of featurization.

Design Takeaways. When performing feature generation from richly formatted data, caching the intermediate results can yield over 1000 \times improvements in featurization runtime without adding significant memory overheads.

4.5.2 Data Representations

The **Fonduer** programming model involves two modes of operation: (1) development and (2) production. In development, users iteratively improve the quality of their labeling functions through error analysis, without executing the full pipeline as in previous techniques such as incremental KBC [26]. Once labeling functions are finalized, the **Fonduer** pipeline is only run once in production.

In both modes of operation, **Fonduer** produces two abstract data structures (**Features** and **Labels** as described in Section 3) that are processed by a data programming engine. These data structures have three access patterns: (1) *materialization*, where the data structure is created; (2) *updates*, which include inserts, deletions, and value changes; and (3) *queries*, where users can inspect the features and labels to make informed updates to labeling functions.

Both **Features** and **Labels** can be viewed as matrices, where each row represents annotations for a candidate (see Section 3.2). **Features** are dynamically named during multimodal featurization, but are static for the lifetime of a candidate; **Labels** are statically named in probabilistic relation classification, but are updated during development. Typically **Features** are extremely sparse: in the ELECTRONICS application, each candidate has about 100 features while the number of unique features can be more than 10M. **Labels** are relatively sparse, where the number of unique labels corresponds to the number of labeling functions.

The data representation that is implemented to store these abstract data structures can significantly affect overall system runtime. In the ELECTRONICS application, multimodal featurization accounts for 50% of end-to-end runtime while probabilistic relation extraction accounts for 15%. We discuss two common sparse matrix representations that can be materialized in a SQL database.

- **List of lists (LIL):** each row stores a list of (column_key, value) pairs. Zero-valued pairs are omitted. An entire row can be retrieved in a single query. However,

updating values requires iterating over sublists.

- **Coordinate list (COO):** rows store (row_key, column_key, value) triples. Zero-valued triples are omitted. With COO, multiple queries must be performed to fetch a row’s attributes. However, updating values takes constant time.

The choice of data representation for **Features** and **Labels** reflects their different access patterns, as well as the mode of operation. During development, **Features** are materialized once, but frequently queried during the iterative KBC process. **Labels** are updated each time a user modifies labeling functions. In production, **Features** access pattern remains the same. However, **Labels** are not updated once the user has finalized their set of labeling functions.

From the access patterns in the **Fonduer** pipeline, and the characteristics of each sparse matrix representation, we find that implementing **Features** as a LIL minimizes runtime both in production and development. **Labels**, however, should be implemented as COO to support fast insertions during iterative KBC and reduce runtimes for each iteration. In production, **Labels** can also be implemented as LIL to avoid the computation overhead of COO. In the ELECTRONICS application, we find that LIL provides 1.4 \times speedup over COO in production, and COO provides over 5.8 \times speedup over LIL when adding a new labeling function.

Design Takeaway. We find that **Labels** should be implemented as a coordinate list during development, which supports fast updates for probabilistic relation extraction, while **Features** should use a list of lists, which provides faster query times. In production, both features and labels should use a list of list representation.

5. EXPERIMENTS

To evaluate the performance of our framework, we conducted experiments in four real-world applications: ELECTRONICS, ADVERTISEMENTS, PALEONTOLOGY, and GENOMICS. Each application contains several relation extraction tasks. The main questions we seek to answer are: (1) how does **Fonduer** compare against both state-of-the-art KBC techniques and manually curated knowledge bases? (2) how does modeling context scopes and multimodality impact quality? and (3) how valuable are noise-aware learning from weak supervision sources in this setting?

5.1 Experimental Settings

Datasets. We use four real-world datasets, one for each of our application domains, which span a variety of sizes and formats. Table 2 shows the statistics of these datasets. We provide more details on each dataset below.

Electronics The ELECTRONICS dataset is a collection of transistor specification datasheets from over 20 manufacturers, downloaded from Digi-Key⁵, a prominent website in the electronic component distribution industry. Specifically, we gathered a corpus of single bipolar transistor datasheets. These documents consist primarily of tables and often express relations via domain-specific symbols. In this application, we extract relations between transistor part numbers and several of their electrical characteristics. We use this

⁵<http://www.digikey.com>

Table 2: Statistics of the datasets used in our experiments.

Dataset	Size	#Docs	#Rels	Format
ELEC.	3GB	7K	4	PDF
ADS.	52GB	9.3M	4	HTML
PALEO.	95GB	0.3M	10	PDF
GEN.	1.8GB	589	4	XML

dataset to evaluate the effectiveness of **Fonduer** on datasets that consist primarily of tables and numerical data.

Advertisements The ADVERTISEMENTS dataset is a collection of webpages which may contain evidence of human trafficking activity. Like many other forms of commerce, trafficking is now online; there are many websites where providers of trafficking services post advertisements containing prices, locations, contact information, physical characteristics, etc. In this application, we extract these attributes for downstream usage by law enforcement. These millions of webpages span 100s of web domains and 1000s of unique layouts. *We use this dataset to examine how robust **Fonduer** is for datasets that contain tremendous data variety.*

Paleontology The PALEONTOLOGY dataset is a collection of well-curated paleontology journal articles on fossils and ancient organisms. In this application, we extract relations between paleontological formations and their corresponding physical measurements. Unlike our other datasets, these papers often have tables that span multiple pages. Thus, achieving high quality in this application requires the ability to link content in tables to the text that references it, which can be separated by 20 pages or more in the document. *We use this dataset to test **Fonduer**'s ability to draw candidates from document-level contexts.*

Genomics The GENOMICS dataset is a collection of open-access biomedical papers on gene-wide association studies (GWAS) from the GWAS Catalog, an extensive quality-controlled, manually curated, collection of studies [30]. In this application, we extract relations between single-nucleotide polymorphisms and human phenotypes found to be statistically significantly associated in the dataset. Note that because this dataset is published in XML format, we do not have visual representations. *We use this dataset to evaluate how well the **Fonduer** framework extracts relations from data that is published natively in a tree-based format.*

Comparison Methods. To evaluate the end-to-end quality of **Fonduer**, we employ two different types of comparison methods: the upper bound of state-of-the-art systems (*Oracle*) and manually curated knowledge bases (*Existing Knowledge Bases*).

Oracle While **Fonduer** uses a unified data model to extract information from all modalities together, other state-of-the-art systems do not. For comparison, we approximate the upper bound of quality of three state-of-the-art information extraction techniques by assuming perfect precision in their extracted relations and experimentally measure their recall.

- **Text** For extraction from text, we follow [16,26]. Candidates are extracted from individual sentences, which are pre-processed with standard NLP tools to add part-of-speech tags, linguistic parsing information, etc.
- **Table** For tables, we follow [2]. Candidates are drawn from individual tables, utilizing cell contents and table

Table 3: End-to-end quality in terms of precision, recall, and F1 score for each applications compared to the upper bound state-of-the-art systems.

Sys.	Metric	Text	Table	Ensemble	Fonduer
ELEC.	Prec.	1.00	1.00	1.00	0.71
	Rec.	0.03	0.20	0.21	0.82
	F1	0.06	0.40	0.42	0.76
ADS.	Prec.	1.00	1.00	1.00	0.88
	Rec.	0.44	0.37	0.76	0.88
	F1	0.61	0.54	0.86	0.88
PALEO.	Prec.	0.00	1.00	1.00	0.92
	Rec.	0.00	0.04	0.04	0.37
	F1	0.00*	0.08	0.08	0.53
GEN.	Prec.	0.00	0.00	0.00	0.92
	Rec.	0.00	0.00	0.00	0.82
	F1	0.00 [#]	0.00 [#]	0.00 [#]	0.87

* Text did not find any candidates.

[#] No full tuples could be created using *Text* or *Table* alone

structure.

- **Ensemble** We also implement an ensemble, as proposed in [8]. Candidates found at the end of both the **Text** and **Table** approaches are merged into a combined candidate set.

Existing Knowledge Base We use existing knowledge bases as another comparison method. The ELECTRONICS application is compared against the transistor specifications published by Digi-Key, while GENOMICS is compared to the both GWAS Central [3] and GWAS Catalog [30], which are the most comprehensive collection of GWAS data and widely-used public datasets. Knowledge bases such as these are constructed using a combination of manual entry, web aggregation, paid third-party services, and automation tools.

Fonduer Details. **Fonduer** is implemented in Python, with database operations being handled by PostgreSQL. All experiments are executed in Jupyter Notebooks on a machine with four CPUs (each CPU is a 14-core 2.40 GHz Xeon E5-4657L), 1 TB RAM, and 12×3TB hard drives, with the Ubuntu 14.04 operating system.

5.2 Experimental Results

5.2.1 Oracle Comparison

We compare the end-to-end quality of **Fonduer** to the upper bound of state-of-the-art systems described in Section 5.1 on four real-world applications. In Table 3, we see that **Fonduer** outperforms the these upper bounds on each dataset. In ELECTRONICS, we see that **Fonduer** improves 70 F1 points over a text-only approach and 34 F1 points over the ensemble in terms of F1 score. In contrast, ADVERTISEMENTS has a higher upper bound with text than tables, which reflects how advertisements rely more on text than the largely numerical tables found in ELECTRONICS. In the PALEONTOLOGY dataset, which depends on linking references from text to tables, the unified approach of **Fonduer** results in an increase of 45 F1 points over the ensemble baseline. In GENOMICS, all candidates are cross-context, preventing both the text-only and the table-only approaches from finding any valid candidates.

5.2.2 Existing Knowledge Base Comparison

We give a detailed comparison between **Fonduer** and existing knowledge bases for ELECTRONICS and GENOMICS. In Table 4, we find that **Fonduer** achieves high coverage of

Table 4: End-to-end quality vs. existing knowledge bases.

System	ELEC.	GEN.	
		Digi-Key	GWAS Central
Knowledge Base			GWAS Catalog
# Entries in KB	376	3,008	4,023
# Entries in Fonduer	445	6,422	6,422
Coverage	0.99	0.82	0.81
Accuracy	0.88	0.87	0.89
# New Correct Entries	17	3,154	2,485
Increase in Correct Entries	1.04×	1.87×	1.43×

the public knowledge bases, while also correctly extracting novel relation entries with over 85% accuracy in both applications. In ELECTRONICS, **Fonduer** achieved 99% coverage and extracted an additional 17 correct entries not found in Digi-Key’s catalog. In the GENOMICS application, we see that **Fonduer** provides over 80% coverage of both existing KBs and finds 1.87× and 1.43× more correct entries than GWAS Central and GWAS Catalog, respectively.

Takeaways. **Fonduer** achieves over 42 F1 points higher quality on average for end-to-end relation extraction compared to the upper bound of state-of-the-art approaches on diverse, real-world datasets. Furthermore, **Fonduer** attains over 80% of existing public knowledge bases while providing up to 1.87× the number of correct entries with high accuracy.

5.3 Inclusion Studies

We conducted inclusion studies to assess the effect of the context scope and multimodal features on the quality of **Fonduer**. In each inclusion study, we change one component of **Fonduer** and hold the others constant. We report the F1 score for each study.

5.3.1 Context Scope Study

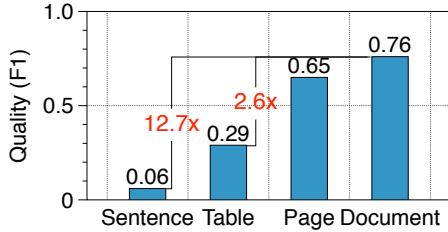


Figure 6: Average F1 score over four relations when broadening the extraction context scope in ELECTRONICS.

To evaluate the importance of addressing the non-local nature of candidates in richly formatted data, we analyze how the different context scopes contribute to end-to-end quality. We limit the extracted candidates to four levels of context scope in ELECTRONICS and report the average F1 score for each. Figure 6 shows that increasing context scope can significantly improve the F1 score. Considering document context gives an additional 70 F1 points (12.7×) over sentence contexts and 47 F1 points (2.6×) over table contexts. The positive correlation between quality and context scope matches our expectations since larger context scope is required to form candidates jointly from both table content and surrounding text. We see a smaller increase of 11 F1

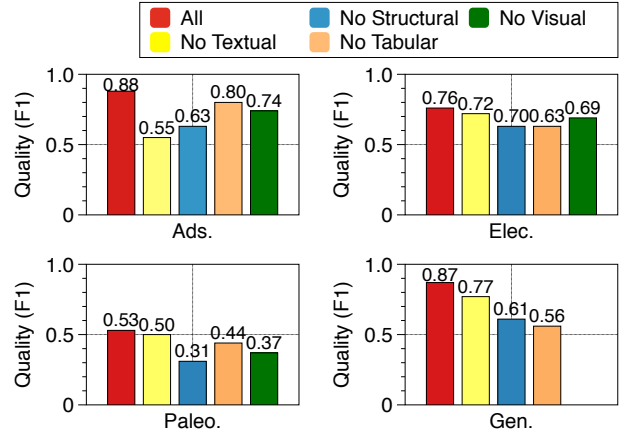


Figure 7: The impact of each modality in the feature library.

points (1.2×) in quality between page and document contexts since many of the ELECTRONICS relation mentions are presented on the first page of the document.

5.3.2 Feature Ablation Study

We evaluate our multimodal feature library, which contains textual, structural, tabular, and visual features. This library eliminates features as an input a user must provide, which significantly reduces development time [33]. We analyze how different features benefit RFIE by comparing the effects of disabling one feature type, while leaving all other types enabled. We report the average F1 scores of each configuration for each application in Figure 7.

From our experiments, we found that removing a single feature set resulted in drops from 3 F1 points (no textual features in PALEONTOLOGY) to 33 F1 points (no textual features in ADVERTISEMENTS). While it is clear in Figure 7 that each application depends on different feature types, we find that it is necessary to incorporate all feature types to achieve the highest extraction quality.

The characteristics of each dataset affect how valuable each feature type is to probabilistic relation classification. The ADVERTISEMENTS dataset consists of webpages which often use tables to format and organize information and many relations can be found within the same cell or phrase. This heavier reliance on textual features is reflected by the drop of 33 F1 points when textual features are disabled. In ELECTRONICS, both components of the (part, attribute) tuples we extract are often isolated from other text (e.g. a lone number within an otherwise empty cell). With little to no textual information to rely on, we only see a small drop of 4 F1 points when textual features are disabled. We see a drop of 22 F1 points when structural features are disabled in the PALEONTOLOGY application due to its reliance on structural features to link between formation names (found in text sections or table captions) to the table itself. Finally, we see similar decreases when disabling structural and tabular features in the GENOMICS application (26 and 31 F1 points, respectively). Because this dataset is published natively in XML, structural and tabular features are almost perfectly parsed which results in similar impacts of these features.

Takeaways. Semantics are often distributed in a document or implied in document structures, and require larger context scope than the traditional sentence-level contexts used

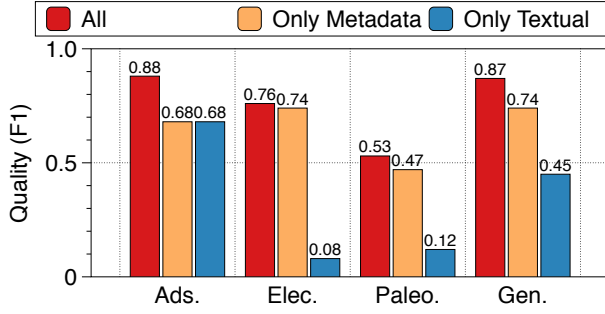


Figure 8: Study of different supervision resources on quality. Metadata includes structural, tabular, and visual information.

in previous KBC systems. Furthermore, it is necessary to utilize all feature types to provide a robust, domain-agnostic description of real-world data.

5.4 Noise Aware Learning Analysis

We evaluate the value of different supervision resources, the impact of modeling noise when learning, and how data programming allows **Fonduer** to benefit from an increased amount of input data.

5.4.1 Supervision Ablation Study

We study how using only textual LFs, only metadata LFs, and the combination of the two sets affects quality. Textual LFs only operate on textual modality characteristics (such as traditional distant supervision rules), while metadata LFs operate on structural, tabular, and visual modality characteristics. Figure 8 shows that applying metadata-based LFs can achieve higher quality than traditional textual-level LFs alone and that the highest quality is achieved when both types of LFs are used. In the **ELECTRONICS** application, we see an increase of 66 F1 points (9.3 \times) when using metadata LFs compared to textual LFs and a 2 F1 point (1.03 \times) improvement over metadata LFs when both types are used. Because this dataset relies more heavily on distant signals, LFs that can label correctness based on column or row header content significantly improve extraction quality. In sharp contrast, the **ADVERTISEMENTS** application benefits equally from metadata and textual LFs. Yet, we increase by 20 F1 points (1.3 \times) when both types of LFs are applied. The **PALEONTOLOGY** and **GENOMICS** applications show more moderate increases of 41 (4.4 \times) and 42 (1.9 \times) F1 points by using both types over only textual LFs, respectively, which reflects how each dataset’s characteristics cater to particular supervision sources. Our experience with **Fonduer** has shown that users are able to effectively express their intuitions using **Fonduer**’s weak supervision resources.

Takeaways. In order to accommodate effective labeling functions, an RFIE framework should preserve information from all of the available modalities and provide easy access to this information in order to leverage higher quality supervision.

5.4.2 Noise Aware Learning Study

We analyze how handling noise in LFs affects the quality of **Fonduer**. **Fonduer** accepts multiple supervision sources so that **Fonduer** can be easily applied to new domains, where existing KBs may not exist. Our learning process models the noise of each source. As a baseline, we compare the

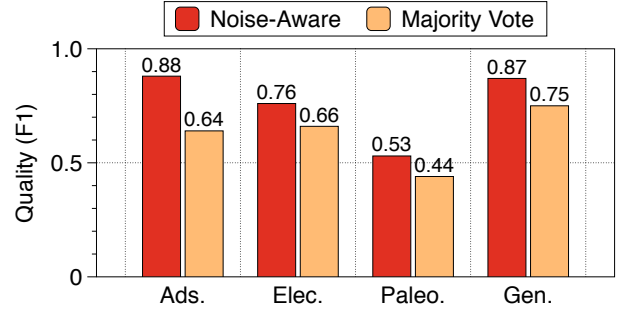


Figure 9: Study of noise-aware learning using data programming (Noise-Aware), as compared to Majority Vote.

noise modeling of **Fonduer** against a majority vote [10] and show the results in Figure 9. In our study, noise-aware learning outperforms the simple majority vote from 9 to 24 F1 points. In **ADVERTISEMENTS**, **Fonduer** achieves significant improvements (24 F1 points) over majority vote, which suggests modeling noise is valuable in datasets with significant data variety. In the less diverse **ELECTRONICS** application, we only see an increase of 9 F1 points, while **PALEONTOLOGY** and **GENOMICS** applications improve by 10 F1 points.

Takeaways. RFIE frameworks should model the noise of each weak supervision source when operating on richly formatted data. Doing so enables a robust programming model that can accept noisy weak supervision sources and still achieve high quality KBC, even in new domains where supervision relies heavily on noisy labeling functions.

5.4.3 Scaling with Quantity of Input Data

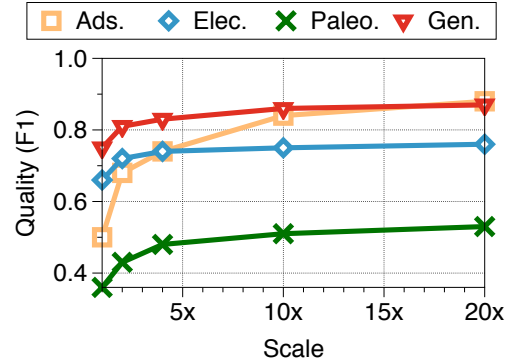


Figure 10: Quality with increasing training set size. The x-axis is the scale of the base number of input documents for each application: **ELECTRONICS** is 50; **ADVERTISEMENTS** is 25,000; **PALEONTOLOGY** is 44; **GENOMICS** is 30.

The process of training models that can take advantage of the breadth of signals provided by our generic feature library requires a prohibitively large amount of training data. We substantiate the effectiveness of user-defined labeling functions in generating this training data by investigating how increasing the number of training documents affects average F1 score. Figure 10 shows how quality changes with increasing training set size. We scale a base number of dataset in

increments of $\{1\times, 2\times, 4\times, 10\times, 20\times\}$ for all four applications while holding other variables constant.

Each of our four applications show quality improvements as the number of input documents is increased. When scaling from $1\times$ to $20\times$, ADVERTISEMENTS quality improves by 38 F1 points ($1.8\times$), while the other applications improve by $1.3\times$ on average. This larger relative improvement may be due to the data variety of the ADVERTISEMENTS dataset which comes from thousands of web domains, whereas the other datasets reflect the lower format variety from academic journals and a small number of manufacturers. Despite this, these applications also benefit from increased input data. In ELECTRONICS, increasing the input data provides **Fonduer** a larger sample of each manufacturer’s style, which improves learning and inference. By using labeling functions, **Fonduer** can avoid cascading inference errors due to lack of supervision that can plague traditional approaches.

6. RELATED WORK

We summarize related prior work in a few categories.

Context Scope KBC systems often restrict extraction context scopes to single sentences [16, 31] or tables [6]. To improve recall, some union results of multiple systems [8, 13], which overlooks many relations such as those crossing unstructured text and tables.

Multimodality Most KBC systems for unstructured data only use textual features [18]. Additional feature types, such as structural features for web tables [22, 24, 29], visual features [12, 32], or combining visual and document tree structure [7, 15], have been proposed to better represent subsets of richly formatted documents.

Supervision Sources Distant supervision has become the de facto technique for creating training data [1, 17, 18, 20]. In addition, crowdsourcing [11] and heuristics from domain experts [21] have also proven to be effective supervision sources.

7. CONCLUSION

In this paper, we study how to extract information from richly formatted data. We show that key challenges of this problem are (1) prevalent document-level relations, (2) multimodality, and (3) data variety. To address these, we propose **Fonduer**, the first KBC system for richly formatted information extraction. We describe **Fonduer**’s unified data model, which enables users to perform candidate extraction, multimodal featurization, and multimodal supervision through a simple programming model. We evaluate **Fonduer** on four real-world domains and achieve an average improvement of 42 F1 points over the upper bound of state-of-the-art approaches. In some domains, **Fonduer** extracts up to $1.87\times$ the number of correct relations compared to expert-curated public knowledge bases.

Acknowledgments.

Thanks to Bryan He for helpful discussions. We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) SIMPLEX program under No. N66001-15-C-4043, the National Science Foundation (NSF) CAREER Award under No. IIS- 1353606, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the Intel/NSF CPS Security grant No.

1505728, the Secure Internet of Things Project, the Sloan Research Fellowship, the Moore Foundation, the Okawa Research Grant, the National Science Foundation Graduate Research Fellowship under Grant No. DGE-114747, the Stanford Finch Family Fellowship, Toshiba, Intel, Google, VMware, Qualcomm, Ericsson, and Analog Devices. We’d also like to thank Prabal Dutta, Mark Horowitz, and Björn Hartmann for their feedback and insights in early versions of this work. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, NSF, ONR, or the U.S. government.

8. REFERENCES

- [1] G. Angeli, S. Gupta, M. Jose, C. D. Manning, C. Ré, J. Tibshirani, J. Y. Wu, S. Wu, and C. Zhang. Stanford’s 2014 slot filling systems. *TAC KBP*, 695, 2014.
- [2] D. W. Barowy, S. Gulwani, T. Hart, and B. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *ACM SIGPLAN Notices*, volume 50, pages 218–228. ACM, 2015.
- [3] T. Beck, R. K. Hastings, S. Gollapudi, R. C. Free, and A. J. Brookes. Gwas central: a comprehensive resource for the comparison and interrogation of genome-wide association studies. *EJHG*, 22(7):949–952, 2014.
- [4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250. ACM, 2008.
- [5] E. Brown, E. Epstein, J. W. Murdock, and T.-H. Fin. Tools and methods for building watson. *IBM Research. Abgerufen am*, 14:2013, 2013.
- [6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.
- [7] M. Cosulschi, N. Constantinescu, and M. Gabroveanu. Classification and comparison of information structures from a web page. *Annals of the University of Craiova-Mathematics and Computer Science Series*, 31, 2004.
- [8] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmman, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610. ACM, 2014.
- [9] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.
- [10] Y. Freund. Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216, 1990.
- [11] H. Gao, G. Barbier, and R. Goolsby. Harnessing the crowdsourcing power of social media for disaster relief. *IEEE Intelligent Systems*, 26(3):10–14, 2011.
- [12] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW*, pages 71–80. ACM, 2007.

- [13] V. Govindaraju, C. Zhang, and C. Ré. Understanding tables in context using standard nlp toolkits. In *ACL*, pages 658–664, 2013.
- [14] M. Hewett, D. E. Oliver, D. L. Rubin, K. L. Easton, J. M. Stuart, R. B. Altman, and T. E. Klein. Pharmgkb: the pharmacogenetics knowledge base. *Nucleic acids research*, 30(1):163–165, 2002.
- [15] M. Kovacevic, M. Diligenti, M. Gori, and V. Milutinovic. Recognition of common areas in a web page using visual information: a possible application in a page classification. In *ICDM*, pages 250–257. IEEE, 2002.
- [16] A. Madaan, A. Mittal, G. R. Mausam, G. Ramakrishnan, and S. Sarawagi. Numerical relation extraction with minimal supervision. In *AAAI*, pages 2764–2771, 2016.
- [17] B. Min, R. Grishman, L. Wan, C. Wang, and D. Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013.
- [18] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.
- [19] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *WSDM*, pages 227–236. ACM, 2011.
- [20] T.-V. T. Nguyen and A. Moschitti. End-to-end relation extraction using distant supervision from external semantic repositories. In *HLT*, pages 277–282, 2011.
- [21] P. Pasupat and P. Liang. Zero-shot entity extraction from web pages. In *ACL (1)*, pages 391–401, 2014.
- [22] G. Penn, J. Hu, H. Luo, and R. T. McDonald. Flexible web document analysis for delivery to narrow-bandwidth devices. In *ICDAR*, volume 1, page 1074, 2001.
- [23] S. E. Peters, C. Zhang, M. Livny, and C. Ré. A machine reading system for assembling synthetic paleontological databases. *PLoS One*, 9(12):e113523, 2014.
- [24] D. Pinto, M. Branstein, R. Coleman, W. B. Croft, M. King, W. Li, and X. Wei. Quasm: a system for question answering using semi-structured data. In *JCDL*, pages 46–55, 2002.
- [25] A. Ratner, C. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. *arXiv preprint arXiv:1605.07723*, 2016.
- [26] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *VLDB*, 8(11):1310–1321, 2015.
- [27] A. Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 2012.
- [28] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.
- [29] A. Tengli, Y. Yang, and N. L. Ma. Learning table extraction from examples. In *COLING*, page 987, 2004.
- [30] D. Welter, J. MacArthur, J. Morales, T. Burdett, P. Hall, H. Junkins, A. Klemm, P. Flicek, T. Manolio, L. Hindorff, et al. The nhgri gwas catalog, a curated resource of snp-trait associations. *Nucleic acids research*, 42:D1001–D1006, 2014.
- [31] M. Yahya, S. E. Whang, R. Gupta, and A. Halevy. ReNoun : Fact Extraction for Nominal Attributes. *EMNLP*, pages 325–335, 2014.
- [32] Y. Yang and H. Zhang. Html page analysis based on visual cues. In *ICDAR*, pages 859–864. IEEE, 2001.
- [33] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *TODS*, 41(1):2, 2016.